October 30, 2001 *LA-UR-01-6222*

Versatile User-Friendly EOS Package

Ralph Menikoff[*] and James J. Quirk[†]
Los Alamos National Laboratory

Abstract

Many application, such as continuum mechanics codes, require material properties as input. Splitting material routines into a separate module, would allow different applications to share and access properties for a wide variety of materials. Here we present an implementation of an equation of state (EOS) package. As an illustration of its use within a hydro code, a high level interface between the EOS package and the Amrita environment is described. This allows a very versatile, user friendly and efficient methodology for setting up and running hydro simulations. In order for a materials package to be of use to a broad community requires some standardization of library functions and protocols for how the material routines interact with other components of a hydro code. To improve material modeling and avoid duplication of effort, the CFD community should strive to develope standards that would facilitate sharing of code.

1 Introduction

Continuum mechanical simulations need to account for material properties. Here we focus on an equilibrium equation of state (EOS) that enters into the formulation of gas dynamics and fluid dynamics. Since there is no single hydro code that is best for all applications, it is useful to have an EOS library to give different codes access to equations of state for a wide variety of materials. The use of an EOS library has three important advantages. First, it avoids duplication of effort among codes. In particular, the addition of a new EOS becomes available to all codes using the library. Second, it facilitates the ability to run a specific application on different codes. Utilizing the same equation of state when comparing different codes avoids systematic errors that would result from the use of different material properties. Third, it is compatible with a modular code structure. Changes to the EOS library do not interact with other aspects of a hydro code.

Though the properties of a given material are unique, different applications use different forms of equation of state (ideal gas, stiffened gas, van der Waal, etc.) according to the region of phase space (density and temperature) that occurs in the application and the needed level of accuracy. Thus, for use in a hydro code, a material requires specifying both the form of EOS and material parameters for the chosen form. An EOS package needs to contain a library of routines for different forms of equation and state, and a database of parameters for a variety of materials. Since different forms of EOS utilize different parameters, an EOS package should also supply auxiliary routines for selecting and initializing a material EOS. In addition, an EOS package should provide ancillary routines to facilitate initializing the material state for a given problem.

It is important to be cognizant of the fact that an equation of state has a domain of validity. For a specified material, the domain is determined in part by accuracy. But more important are thermodynamic properties required for the stability of the solution to the initial value problem of the PDEs that are the basis for hydrodynamic simulations. When an EOS violates the stability condition, even the best hydrodynamic

[*]T-14, `rtm@lanl.gov`
[†]CCS-2, `quirk@lanl.gov`

algorithm will have problems. In other words, a material equation of state should be regarded as input to a hydro code, and bad input will result in bad output.

The domain of an EOS can be determined by checking thermodynamic consistency and other thermodynamic relations [8], such as the sound speed being real and positive, needed to guarantee that fluid flow has a physically reasonable wave structure. An EOS package should provide routines to determine the valid domain of a particular equation of state. For reasons of efficiency, it is best to run the thermodynamic tests as a preprocessing step separate from any hydro code.

The ability to access an EOS database outside a hydro code is useful for purposes other than checking thermodynamic properties. It provides a convenient means for calibrating EOS parameters to data, and for comparing different fitting forms for a particular material. Most of the available high pressure data comes from shock wave experiments. Computing points on the Hugoniot locus and release isentrope are very useful in designing and analyzing such experiments. It is highly desirable for an EOS package to provide an interface to a database that allows a user to conveniently determine such properties.

An outline of the remaining sections follows. Section 2 describes the structure of an EOS package that provides the capabilities discussed above. It is designed to be versatile by including high level routines to allow new materials and new forms of equations of state to be added easily. Section 3 describes a graphical interface. This allows easy access to the EOS database, and the determination of the most relevant thermodynamic quantities for fluid flow. Section 4 describes the implementation of the EOS package in a one-dimensional multi-material hydro code. For this purpose the Amrita environment of James Quirk [11, 12] is used. Amrita contains an extendable scripting language. Utilizing the parser capabilities of Amrita allows for a simple user-friendly setup of hydro simulations. Section 5 discusses extensions to other material properties within the same general framework employed by the EOS package.

The specific EOS routines and their use within a hydro code discussed here is meant to show that a versatile, user-friendly EOS package advocated here can indeed be implemented. The source code is available upon request. But the main intent of this article is to give the computational fluid dynamics community ideas for improving software related to material properties and the setup of hydro simulations pertaining to the selection of materials and the initialization of hydrodynamic states. Related remarks are presented in the last section.

2 Structure of EOS package

The EOS package is split into three parts: databases, an application interface and shared libraries. A database is a file containing the parameters that define an equation of state for specific materials. The application interface is a collection of routines that need to be linked with an application using the EOS package. It defines an EOS as a family of pointers to functions that return various material quantity, such as a function `P(V,e)` that returns the pressure. The shared libraries are dynamically linked and contain the low level routines that actually compute the material quantities for a specific type of EOS.

An application obtains an EOS by passing the application interface a string of the form ‘type::name’, where ‘type’ specifies the form of equation of state and ‘name’ specifies the particular substance. The application interface reads the databases, fetches an EOS of the desired type, initializes the EOS with parameters from the database and loads the needed shared library. Applications utilize material properties indirectly through the pointers to the EOS functions provided.

This structure is very versatile. An equation of state can be defined by an analytic form, implicitly as the solution to an auxiliary set of equations or as a table lookup. A new material of an existing type can be added simply by including the material parameters in a database. For a material of a new type, first the shared library corresponding to that type of equation of state is generated, and then the material parameters are added to a database. The database also contains the name of the shared library corresponding to each type of equation of state. The net result is that new materials can be added to the database and used by an application without changing or even relinking the application.

This structure is particularly advantageous for dealing with proprietary or classified material properties. Classified data can be placed in a separate database file and the corresponding routines defining the form of the equation of state placed in a separate shared library. Thus, there is a very a clear and clean mechanism to separation proprietary information from the general functions of the EOS package which are public.

Next, each part is described in more detail.

2.1 Database

A database is an ASCII file. It contains a data block for each material. The data block has the form given in the table 1 for the material `Hayes::HMX`. The data block is simply a list of pairs of strings of the form `parameter = value`. The EOS package provides routine that strip out comments and transform each data block into a canonical form that is easy for an EOS initialization routine to process. The interpretation of the (parameter,value) pairs is left up to the initialization routine for the specific type of EOS. Thus, the purpose of the database routines is to translate the data file from a user-friendly form to a machine efficient form. This greatly simplifies writing initialization routine for each EOS type.

The EOS package also provides a calculator that allows initialization routines easily to evaluate expressions. In the above example, it is more convenient to specify the initial density then the specific volume (`V = 1/1.9` instead of `V = 0.526313`). It is also more meaningful to specify the energy with an algebraic expression rather than with a numerical value (`e0 = T*Cv` instead of `e0 = 0.45`).

The database provides two additional services. First, it keeps track of units. The units are specified within the database. In the previous example, the system of units `hydro.std` is defined by the three data blocks listed in table 2. The EOS package also provides routines to facilitate changing between systems of units. Changing units by hand is very error prone. But it is simply bookkeeping that is straight forward to automate on a computer. The units example illustrates that it is straight forward to construct composites from other data blocks. This feature can be used to good affect. For example, an EOS of a mixture can be constructed from the equations of state of the components together with a mixture rule, such as pressure and temperature equilibrium.

Table 1: Data block for the material `Hayes::HMX`

```
Hayes=HMX; units=hydro.std
{
   /* parameters fit to Olinger’s hydrostatic data */
   V0 = 1/1.9    # g/cm^3
   P0 = 1.0e-4   # GPa
   T0 = 300      # degrees K
   K0 = 13.5     # GPa
   N  =  9.3
   Cv = 1.5e-3   # (MJ/kg)/K
   Gamma0 = 1.1
   e0 = T0*Cv
   #c0=2.6778 s = 2.68946
}
```

Table 2: Data blocks for the units `hydro.std`

```
Units=hydro.std
{
   Values = hydro.std:values
   Names  = hydro.std:names
}
Values=hydro.std:values
{
# Fundamental
   L = 1   # length
   t = 1   # time
   m = 1   # mass
   T = 1   # temperature
   :use = hydro.derived:values
}
Names=hydro.std:names
{
   length          = mm
   time      = micro‘s
   mass      = mg
   temperature   = K
   #
   force      = kN
   energy      = J
   #
   velocity   = km/s      # km/s = mm/micro‘s
   density      = g/cm^3   # g/cm^3 = 10^3 kg/m^3
   specific_volume   = cm^3/g
   specific_energy   = MJ/kg
   pressure   = GPa      # GPa = 10 kb
   specific_heat   = MJ/kg/K
   :use = hydro.derived:names
}
```

Table 3: Data block for the material Hayes::HMX1 illustrating the use of the database construct :use.

```
Hayes=HMX; units=hydro.std
{
   :use=HMX
   /* parameters fit to Yoo-Cynn’s hydrostatic data */
   K0 = 12.4     # GPa
}
```

The second point is the construct `:use = hydro.derived:names`. This allows the user to easily modify an existing EOS. For example, a second EOS for HMX with a different value of the bulk modulus than that used by the EOS `Hayes::HMX` can be specified with the data block in table 3. This is particularly useful for studies of the sensitivity of an experiment to uncertainties in EOS parameters.

2.2 Application interface

The implementation of the EOS package is based on an object oriented approach. It is written in C++. The hydro code described in the next section predates the EOS package and is written in FORTRAN. There is no difficulty in mixing languages.

The interface for an EOS is defined as an abstract base class. By utilizing only the virtual functions in the base class, an application can treat all types of equations of state in the same manner. Specific forms of equations of state are derived classes. A derived class contains the parameters for a specific material and implements the virtual functions.

Only two virtual functions are required; an initialization routine and a pressure function. Generic routines are provided for other quantities. For example, the generic routine for sound speed performs a numerical differentiation, while the generic routine for an isentrope invokes an ODE solver, and the generic routine for a shock wave solves the Hugoniot equation. The intent is that the derived class overlays the generic functions with more efficient and accurate routines that take into account the particular form of equation of state that it implements. The generic routines serve two purposes. First, very little code needs to be written in order to try out a new form of EOS. If the new form works satisfactorily, the remaining routines can be added. On the other hand, if the new form is not satisfactory then the time spend to test it out is minimal. The second purpose of the generic routines is in validating a new EOS. They may be inefficient but they do provide reasonably accurate values for quantitative comparisons.

Isentropes, Hugoniots and isotherms are important families of curves that arise in hydrodynamics. Through any given point there is a unique curve of each type. It is natural to implement these loci as classes. The EOS functions for these quantities take a state as an argument and return a pointer to a class for the corresponding quantity appropriately initialized for the locus through the specified state. The isentrope, Hugoniot and isotherm classes have a generic interface providing functions that determine the state on the locus at specified values of pressure or specific volume or other hydrodynamic quantities relevant to the loci. For example, a point on the Hugoniot locus also can be specified in terms of the particle velocity or shock velocity. The loci classes give the flexibility to overlay generic routines with more efficient specialize routine for particular types of equations of state.

For this setup to be effective, it is necessary that the equation of state class is reference counted. This allows the loci classes to store a pointer to the EOS and avoid copying the EOS parameters. In the case of a tabular EOS, a large amount of storage may be used. Reference counting also allows the EOS package to catch a common programing error, deleting an EOS while other classes still use it. It is particularly time consuming to track down allocation errors since the code typically crashes at a point remote from the cause of the problem.

A function of importance to hydrodynamics is a Riemann solver. The isentrope and Hugoniot classes greatly facilitated the writing of the generic Riemann solver provided by the EOS package. It is natural for the solution algorithm to iterate on pressure or particle velocity. On the other hand, the isentrope is defined by an ODE in which the natural independent variable is the specific volume. The classes provide an interface that allows the most convenient variable to be used. In addition, the ODE solver, used by the generic isentrope class, caches points as it performs the integral. This avoids repeated integration of the same portion of the isentrope during the iteration algorithm and greatly improves the efficiency of the Riemann solver.

In summary, the application interface has three purposes. First is to provide a common interface for all types of equations of state. Second is to relieve users the burden of low level considerations such as storage allocation and efficiency. Third is to avoid duplication of effort by providing general routines, such as those in the loci classes and the Riemann solver, so that different users don't repeatedly write similar routines. Other general purpose routines can be build, as needed, upon the routines currently in the EOS package. For example, the two-dimensional analog of the Hugoniot locus and the Riemann problem are shock polars and wave patterns. This capability would be useful for some applications. In addition, the Riemann solver in the EOS package is limited to convex equations of state. Generalizing to the non-convex case is possible [9] and would be very useful for analyzing shock induced phase transitions. Adding such routines to the package would make them available to all users. Thus, the EOS package should not be thought of as complete but rather as an ongoing work that serves the needs of those who chose to use it.

2.3 Shared libraries

Presently the EOS package includes five types of equations of state. Ordered by level of complexity these are:

1. The **ideal gas** EOS is defined by the formula

$$P(V,e) = (\gamma - 1)e/V \tag{1}$$
$$T(V,e) = e/C_v \tag{2}$$

   where γ is the adiabatic index and C_v is the specific heat at constant volume. Simple analytic formulas are available for the isentrope, Hugoniot and isotherm loci. The implementation of this case illustrates the use of overlaying generic functions with efficient and accurate routines that takes advantage of the particular form of EOS.

   Even in this simplest of cases an analytic solution to the Riemann problem does not exist. A Riemann solver specialized for an ideal gas is about 3 times faster than the generic one for a typical problem. The generic solver is slower when the solution requires a longer integration along the isentrope. It would be considerably slower if the generic Hugoniot and isentrope classes were used instead of the specialized ones for an ideal gas.

2. The **stiffened gas** EOS is defined by the formula

$$P(V,e) = (\gamma - 1)(e - e_*)/V - P_* \tag{3}$$
$$T(V,e) = (e - P_* V - e_*)/C_v \tag{4}$$

   This is a variation on the ideal gas that allows an extra parameter to set the initial sound speed, $c_0^2 = \gamma(P_0 + P_*)V_0$. Physically, it corresponds to translating the isentrope in the (V,P)–plane up in pressure by an amount P_*. The isentrope is frequently refered to as the Tait EOS.

3. The **Hayes** EOS [4] is defined by the free energy

$$\begin{aligned} F(V,T) = e_0 + P_0(V_0 - V) - \eta_0 T \\ + C_v(T - T_0)\left[1 + \frac{\Gamma_0}{V_0}(V_0 - V)\right] - C_v T \ln(T/T_0) \\ + \frac{K_0 V_0}{N(N-1)}\left[(V_0/V)^{N-1} - (N-1)(1 - V/V_0) - 1\right], \end{aligned} \tag{5}$$

   where V_0, P_0, T_0, e_0 and η_0 denote the specific volume, pressure, temperature, specific energy and entropy at a reference state, C_v is the specific heat at constant volume, Γ_0 characterizes the Grüneisen coefficient, $\Gamma/V = \Gamma_0/V_0$, and the parameters K_0 and N characterize the isothermal bulk modulus, $K_T = K_0(\rho/\rho_0)^N$. In this case there are analytic formula for the isentropes and isotherms but not for the Hugoniot locus. Consequently, the generic Hugoniot class is used for this case.

With appropriate parameters, the Hayes EOS is very similar to a Mie-Grüneisen EOS together with a linear relation, $u_s = c_0 + s u_p$ for the principal Hugoniot. The Mie-Grüneisen EOS is frequently used to model metals, and the parameters c_0 and s have been measured for many materials [5]. There are simple relations between the values of c_0 and s and the parameters K_0 and N. Because of its usefulness, the initialization routine was written to accept either the pair K_0 and N, or c_0 and s. This example illustrates the advantage of allowing each type of EOS to have control over its initialization.

4. The **von Mises Elastic-Plastic** EOS is a simple generalization of a hydrostatic EOS for the case of uniaxial strain. The stress is defined in terms of a hydrostatic EOS, $P_s(V, e)$ by

$$P(V, e) = P_s(V, e) + \tfrac{4}{3} G\epsilon\,, \tag{6}$$

where G is the shear modulus, the elastic strain is given by $\epsilon = \max[\log\left(\frac{V_0}{V}\right), \frac{Y}{2G}]$ and Y is the yield strength.

The elastic-plastic transition displayed in the Hugoniot is similar to that arising from a phase transition. Moreover, this EOS illustrate how a composite EOS can be constructed from a pre-existing one motivated by an enhancement of the underlying physical model.

5. The **equilibrium porous** EOS is a semi-analytic thermodynamic consistent complete EOS [7]. The pressure is defined in terms of a pure phase EOS, $P_s(V, e)$ by

$$P(V, e) = \phi P_s(\phi V, e - B(\phi))\,, \tag{7}$$

where the volume fraction ϕ is determined implicitly by the equation

$$P(V, e)V = \phi \frac{dB}{d\phi} \tag{8}$$

and the compaction energy $B(\phi)$.

The compaction law, $B(\phi)$ and $\frac{dB}{d\phi}$, is encapsulated in a class. This allows the same database mechanism as used by the **EOS** class and the **Units** class to be used to fetch and initialize the compaction law needed by the **equilibrium porous** EOS.

These EOS types illustrate that the framework of the EOS package is quite general and allows for a wide variety of forms of equations of state. The package is very modular. The addition of a new shared library would not affect any existing code within the package. Since the database routines are designed to accept multiple data files, existing data files do not have to be modified. Once the shared object and the data file are written, the use of a new material by an application requires only the input of the name of the new material and the name of the data file that specifies its parameters.

3 Graphical Interface

To test the EOS package several driver routines were written. These are high level programs which take command line input and then exercise routines within the package. Though somewhat clumsy to use because of the inflexible nature of the command line arguments, the functions they perform are generally useful. These include programs to fetch data from the database, to compute points on the loci of an isentrope, Hugoniot or isotherm, and to solve for the impedance match of a shock wave propagating through the interface between any pair of materials.

To make it more convenient to use these programs, a graphical interface is included in the EOS package. The interface is written in perl/Tk. (A more portable version would be written in Java and run on a web browser. The interface has been partly converted by a summer student, Peter Keeler.) The interface allows the user to enter data in a convenient form, calls the EOS programs to do the calculations and then displays the results in a readable form. Figure 1 shows the user window that the graphical interface generates. The interface displays units along with numerical values for hydrodynamic quantities. In addition, it can produce lists of points or plot pairs of variables along an isentrope or Hugoniot locus. It also can generate the graphic solution to an impedance match problem. An example is shown in figure 2.

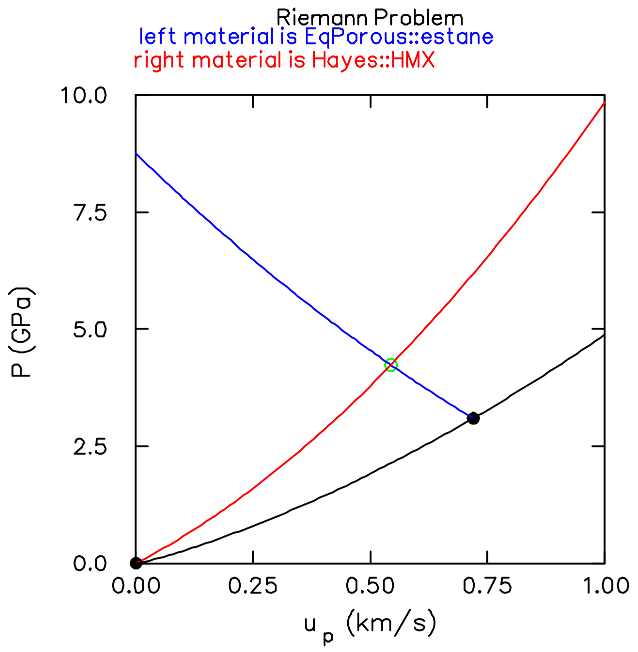
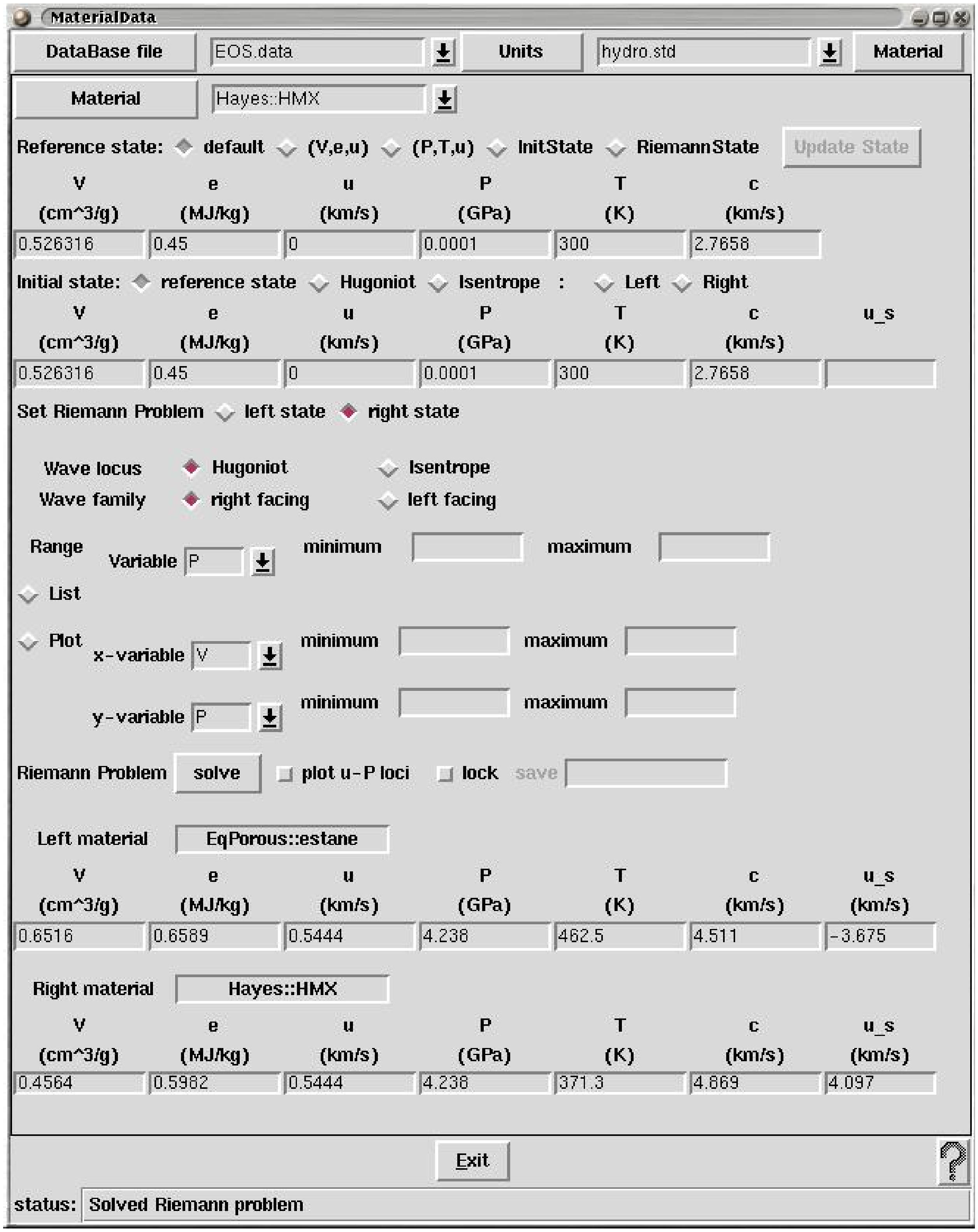


Figure 2: Graphical solution to impedance match problem generated using the graphical interface to the EOS package. In this case, the solution has two outgoing shock waves.

4 Utilizing EOS package within Hydro code

Utilizing the EOS package in a hydro simulations is described next. The Amrita environment developed by James Quirk [11] is used because the flexibility of its structure is well suited for the task. The purpose is to show that it is straight forward to implement a code with the material properties and the hydro algorithm in separate modules, and to demonstrate a concise user-friendly setup procedure for hydro problems.



DataBase file: EOS.data
Units: hydro.std
Material: Hayes::HMX

Reference state: [✓ default] (V,e,u) (P,T,u) InitState RiemannState

V (cm^3/g)	e (MJ/kg)	u (km/s)	P (GPa)	T (K)	c (km/s)
0.526316	0.45	0	0.0001	300	2.7658

Initial state: [✓ reference state] Hugoniot Isentrope : Left Right

V (cm^3/g)	e (MJ/kg)	u (km/s)	P (GPa)	T (K)	c (km/s)	u_s
0.526316	0.45	0	0.0001	300	2.7658	

Set Riemann Problem left state [✓ right state]

Wave locus [✓ Hugoniot] Isentrope
Wave family [✓ right facing] left facing

Range Variable P minimum maximum
List
Plot x-variable V minimum maximum
y-variable P minimum maximum

Riemann Problem solve plot u-P loci lock save

Left material: EqPorous::estane

V (cm^3/g)	e (MJ/kg)	u (km/s)	P (GPa)	T (K)	c (km/s)	u_s (km/s)
0.6516	0.6589	0.5444	4.238	462.5	4.511	-3.675

Right material: Hayes::HMX

V (cm^3/g)	e (MJ/kg)	u (km/s)	P (GPa)	T (K)	c (km/s)	u_s (km/s)
0.4564	0.5982	0.5444	4.238	371.3	4.869	4.097

status: Solved Riemann problem

Figure 1: Graphical interface to EOS package.

4.1 Amrita

Amrita is an environment that greatly facilitates setting up, running and documenting the results of hydro simulations. It is composed of several parts; scripting language, adaptive mesh refinement computational engine, documentation facility, and an extensive library of useful procedures and tools. It is designed to be an extendible system. This is an important feature that we use to take full advantage of the capabilities of the EOS package.

A brief overview of Amrita is needed to understand how we will use it. We call the scripting component of Amrita the front end. Upon startup, the front end initiates a separate process, which we call the back end, along with a two way communication channel called the ISL pipe. A computational engine is then plugged in by dynamically linking a shared object to the back end. The essential idea is that the front end schedules work for the computational engine on the back end to carry out.

An amrita script is processed by an interpreter. This allows tasks, such as setting up and running a simulation, to be specified in a concise user-friendly manner. The computer time expended processing a script is negligible compared to the computational effort running a hydro simulation. Scripts also provide a means of automating and documenting the steps taken to analyze an application, from setting up a calculation to postprocessing the results. This is particularly convenient when the need arises to change a parameter and rerun a simulation.

The first step of the interpreter is to read and parse a line, translating the high level script commands into low level instructions of an intermediate scripting language (ISL). After sending the instructions down the ISL pipe, the interpreter then waits for the computational engine to send back a reply. Upon receiving a reply that the command has been completed, the interpreter procedes to process the next line of the script. This description of the action of the interpreter is oversimplified. Like processors of other scripting languages, the amrita interpreter also has the capability to set and use tokens, to perform loops, to call other amrita scripts as procedures, and to execute shell commands or system calls. Moreover, the amrita interpreter can utilize script fragments in other languages such as perl.

The back end can be thought of as event driven. In contrast to a window system, the events are generated by the parser at the front end rather than mouse clicks or keystrokes. By splitting the steps carried out by the computational engine into small tasks, each corresponding to an ISL instruction, one gains the ability to program the computational engine for a specific application rather than having a hardwired program. That is to say, one doesn't have to add problem dependent code and recompile the computational engine in order to add a specialized feature needed to run a particular problem.

We use amr_sol, also written by James Quirk [12], for the computational engine. It provides the infrastructure needed for an adaptive-mesh-refinement hydro code. To amr_sol we plug in a solver or patch integrator. The solver determines both the equation set and the hydro algorithm. Its function is to update the solution vector on a simple rectangular grid by one time step. Amr_sol takes care of the IO, boundary conditions, memory allocation and bookkeeping associated with refining the grid, filling in ghost cells for each grid patch, and subcycling over the finely zoned grid patches. We note that the data structures needed to split a grid into patches for mesh refinement are also well suited for parallelizing a hydro simulation. Amr_sol is designed to take advantage of this and can be run on either a single processor or a cluster of work stations.

In addition to plugging in a computational engine and a solver, an EOS plugin was written to interface the parser at the front end and the solver at the back end with the EOS package. As discussed in the previous section, the EOS package dynamically links libraries with the specialize EOS types that it needs. The use of a hierarchical sequence of plugins or dynamically linked libraries is a very powerful technique that allows for great flexibility in expanding capabilities of a code as needs arise.

Although amr_sol can accommodate two-dimensional grids, for the purpose of demonstrating the use of the EOS package in a multi-component hydro code, we restrict our attention to one-dimensional solvers for the Euler equations in Lagrangian coordinates

$$\frac{\partial}{\partial t}\begin{pmatrix} n \\ V \\ u \\ E \end{pmatrix} + \frac{\partial}{\partial m}\begin{pmatrix} 0 \\ -u \\ P_n \\ P_n u \end{pmatrix} = \vec{0}\,, \tag{9}$$

where n is a material index, V is specific volume, u is particle velocity, $E = e + \frac{1}{2}u^2$ is the total specific energy, $P_n(V, e)$ is the pressure for the n^{th} material and $dm = \rho dx$ is the mass coordinate. A Lagrangian algorithm naturally tracks contacts and avoids the complication with an Eulerian algorithm that results from mixed cells. This is done for simplicity and is not a fundamental limitation.

The solver or patch integrator is a small amount of logically simple code compared to the adaptive-mesh-refinement infrastructure provided by amr_sol. Four solvers were written; viscous solver, Roe solver [13] as formulated by Glaister [2, 3] for general EOS, Lax-Friedrichs solver and Marquina solver [1]. These are basically variations on a theme and differ principally in the form of numerical dissipation needed for shock capturing. The solvers interact with the EOS package through the use of binding functions. This indirection is needed to allow the FORTRAN functions of the solver to access the C++ classes of the EOS package.

We note that the Roe scheme uses an average state of adjacent cells. This is suitable for the interior of a region but not at the interface between two components with very different material properties. At an interface the Roe scheme is replaced with a Riemann solver. When the difference in the values of u and P across the interface are small, a linearized Riemann solver is used. Otherwise, the exact Riemann solver provided by the EOS package is used. Since the exact Riemann solver is applied for a very small fraction of cells in the grid, the extra expense has a very small effect on the overall efficiency of the solver. Having routines, such as a Riemann solver, available in the EOS package, allows special cases to be handled in a rational and robust manner, and alleviates the need for ad hoc fixes.

4.2 EOS plugin

A plugin requires two parts working in concert. When a plugin is started, it registers a set of keywords with the front end and a set of ISL tags with the back end. The amrita interpreter, upon recognizing a keyword, calls the procedure that the plugin registered. The procedure for the plugin parses the remainder of the script line, and sends instructions for the desired task through the ISL pipe to the back end. Similarly, the back end upon recognizing an ISL tag calls the subroutine that the plugin registered. Both ends of the plugin have the ability to read from and write to the ISL pipe. This enables data to be transferred between the interpreter and the plugin. Moreover, the intermediate scripting language is not etched in silicon, but is programmable. In effect, the ISL is determined by the tags and whatever tasks the plugins are designed to perform.

The most important part of developing a plugin is to select keywords that are sufficiently flexible to allow the user to specify any reasonable task, yet clear and concise for the idioms or patterns of work most frequently used. The keywords for EOS plugin are listed in table 4. The term `handle` is just a variable name and is a convenient means of referring to an EOS data structure. It can be passed on to other EOS keywords to obtain information about an EOS state or specify the input for a function, such as computing a point on a Hugoniot locus or solving a Riemann problem.

Table 4: EOS plugin keyword commands

getmaterial	type::name	-> handle
getstate	on [dir] locus(handle) at var = num	-> handle
	where dir = left | right	
	locus = hugoniot | isentrope | isotherm	
	var = P | V | u | us	
getstate	handle at Ve | PT expr_list	-> handle
	where expr_list has form: var = expr; . . .	
solve	RiemannProblem (left_state, right_state)	-> riemann:handle
eosinfo	index <type>	-> token_name
	<type> = materials | units | type	
eosinfo	type <type>	-> token_name
	<type> = *::<name> | <eos_type>	
eosinfo	units <name> (variable_list)	-> namespace::|prefix
eosinfo	convert <to_from> (var_list)	-> namespace::|prefix
	<to_from> = to <units> | from <units> | <units1> to <units2>	
eosinfo	material <name>	-> token_name
eosinfo	state::<handle> (var_list)	-> namespace::|prefix
eosinfo	rp::<handle>:<hash> (var_list)	-> namespace::|prefix
	<hash> = left_state | right_state | left_wave | right_wave | units	
eosinfo	[state] <handle> statement_list	-> namespace::|prefix
locus	{ # multi-line keyword	
	print {var[(format)], var[(format)], ...}	
	on [dir] locus(handle)	
	for var {init; logic_expr; increment}	
	}	

As an illustration of the use of keywords in proscribing a hydro problem, the Amrita script commands needed to setup the impedance match test problem corresponding to figure 2 are

```
def SolutionField
    getstate on right hugoniot(EqPorous::estane) at P=3.1  -> W'left
    getmaterial Hayes::HMX  -> W'right
    setfield W'left  X[] <  $Nc
    setfield W'right X[] >= $Nc
end def
```

This sets the left and right materials to be `EqPorous::estane` and `Hayes::HMX`, respectively. Furthermore, the state `W'left` is set to the point on the principal shock Hugoniot with a pressure of 3.1, and `W'right` is set to the reference state defined in the database, which typically corresponds to $P = 1$ bar and $T = 300$ K. The `SolutionField` block is an interlock to facilitate error handling by ensuring portions of the setup are perfomed in the correct order. In this example, `setfield W'left X[] < $Nc`, would not make sense unless the grid had been defined previously.

The ability to set a state on a Hugoniot is very useful. In many material property experiments, high pressures are obtained from either a gas gun projectile or an explosive plane wave lens. Frequently, the drive system is not of interest and can be replaced by the lead shock that it generates. In the case of a gas gun experiment, the velocity of the projectile is measured, and the lead shock wave in the target is determined by a Riemann problem. The shock state in the target can be set with the script commands

```
getstate Hayes::Kel-F at Ve{ u = $u_projectile } -> projectile
getmaterial Hayes::Quartz.x-cut -> target
solve RiemannProblem(projectile,target) -> drive
set target_state #= &eos::hydro_state("rp:drive::right_wave")
W’target ::= <$target_state>
```

Determining the state within a script setting up a simulation avoids the need for a side calculation and the error prone step of transcribing data to the input file of the hydro code.

The effort of writing the EOS plugin is comparable, in terms of number of lines of code, to that of writing the graphical interface for the EOS package. We note that figure 2 obtained with the graphical interface can also be generated with the script commands

```
getmaterial EqPorous::estane                 -> estane
getstate on right hugoniot(estane) at P=3.1 -> incident
getmaterial Hayes::HMX                       -> hmx
solve RiemannProblem(incident,hmx)           -> RP
PlotRiemannSolution {
  handle   = RP1
  incident = estane
  extend   = true
   u_max   = 1.0
}
```

The command `PlotRiemannSolution` is an amrita procedure. It uses the keyword `locus` to obtain the data needed for the plot. Another library procedure `ProfileRiemannSolution` plots the profile of a specified variable at a specified time. This procedure is used in the next section to provide a comparison for test problems of a hydro code.

For the casual user, the graphical interface is easier to learn and more convenient then writing a script. However, the scripting language is more versatile and powerful. The above code fragments and procedures give an indication of the utility and flexibility that the keywords provide. Moreover, scripts greatly facilitate documentation and automation. Scripts are well suited to running a series of calculations, such as testing the thermodynamic consistency of all materials in a database, or comparing different fitting forms for the same material.

4.3 Numerical examples

As a test of the implementation of the EOS package in a hydro code, we use a Riemann problem. In the case with ideal gas equations of state this test is known as Sod’s problem [14]. Though there is no analytic solution, the theory of hyperbolic PDEs with scale invariant initial data allows any Riemann problem with general equations of state to be solved arbitrarily accurately. This effectively provides an “exact” solution to compare with the results from a hydro code. The use of materials with disparate properties, large ratio of densities and sound speeds, provides a more stringent test for hydro algorithms than the Sod problem.

We consider impedance match problems. The moment the incident shock impacts the material interface corresponds to the initial conditions for a Riemann problem. The first test problem has a solution, shown in figure 2, consisting of two outgoing shock waves. The numerical profiles of various quantities are compared with the exact solution in figure 3. Aside from smearing out an ideal discontinuous shock front, as expected for a shock capturing algorithm, the numerical solution is quite accurate.

Without mesh refinement adapted to follow the shock fronts, the solution is shown in figure 4. Two points are noteworthy. First, without mesh refinement, the shock fronts are quite smeared out. Second, there is a significant overshoot in the energy at the contact. The smearing of the shock fronts is related to a property of the equations of state, namely, a large sound speed relative to the shock speed. For both the weak reflected shock and the strong transmitted shock, the increase in characteristic velocity across the shock front is small. Hence, the convergence of the characteristic, which leads to a shock, is not as strong an effect as with ideal gases. The energy anomaly at the contact is similar to the ‘excess wall heating’ that occurs in the Noh problem [10]. It results from an entropy error during the transient when the reflected shock forms [6]. In contrast to an Eulerian algorithm, there is no advection in an Lagrangian algorithm to smear the entropy error over many cells. By coarsening the local mesh behind the shock front, adaptive mesh refinement has a diffusive effect behind the shock front.

As a second example, we use a test problem in which the reflected wave is a strong rarefaction. The wave curves for the graphical solution are shown in figure 5. A comparison of the numerical profiles with the exact solution and the effect of using adaptive mesh refinement is shown in figures 6 and 7. With adaptive mesh refinement, the leading and trailing edge of the rarefaction, which represent kinks or points with discontinuous first derivative, are much better resolved. Both cases show an energy anomaly at the contact. It is due to the formation of a centered rarefaction. The rarefaction can not be resolved until it is spread out over several cells. The lack of resolution causes an entropy error. For stiff materials, the entropy error causes a much larger anomaly in the energy than in the density.

The use of scripts and an EOS database greatly facilitates the automation of test problems. This is important because non-linear effects on numerical algorithm for systems of equations are beyond the current techniques for mathematical analysis. Consequently, a large number of test problems are needed to assess the strengths and weaknesses of the available algorithms. The ability to plugin a solver allows different algorithms easily and systematically to be compared. Moreover, with all other factors being identical, the effects of the algorithm are isolated clearly.

5 Extensions to other Material Properties

Two classes of problems, reactive flow and elastic-plastic flow, are natural generalizations of the Euler equations. Both require additional material properties. Again it would be advantageous for a code to obtain the needed properties from a database. We briefly outline how material properties in general can be incorporated into a database using the same framework as employed here for equations of state.

Though the physical description of reactive flow and elastic-plastic flow are different, the mathematical structure is the same. For simplicity we consider elastic-plastic flow for the case of uniaxial strain, and reactive flow for the case in which all species are in pressure and temperature equilibium. Both cases are then extensions to the Euler equations obtained by adding internal degrees of freedom with corresponding governing equations, which have the form of rate equations,

$$\frac{\partial}{\partial t}\begin{pmatrix} n \\ V \\ u \\ E \\ \vec{Z} \end{pmatrix} + \frac{\partial}{\partial m}\begin{pmatrix} 0 \\ -u \\ P_n \\ P_n u \\ \vec{0} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ Q \\ \vec{R} \end{pmatrix} , \tag{10}$$

where $\vec{R}$ and Q are source terms. The rate equations are in characteristic form. Although they increase the

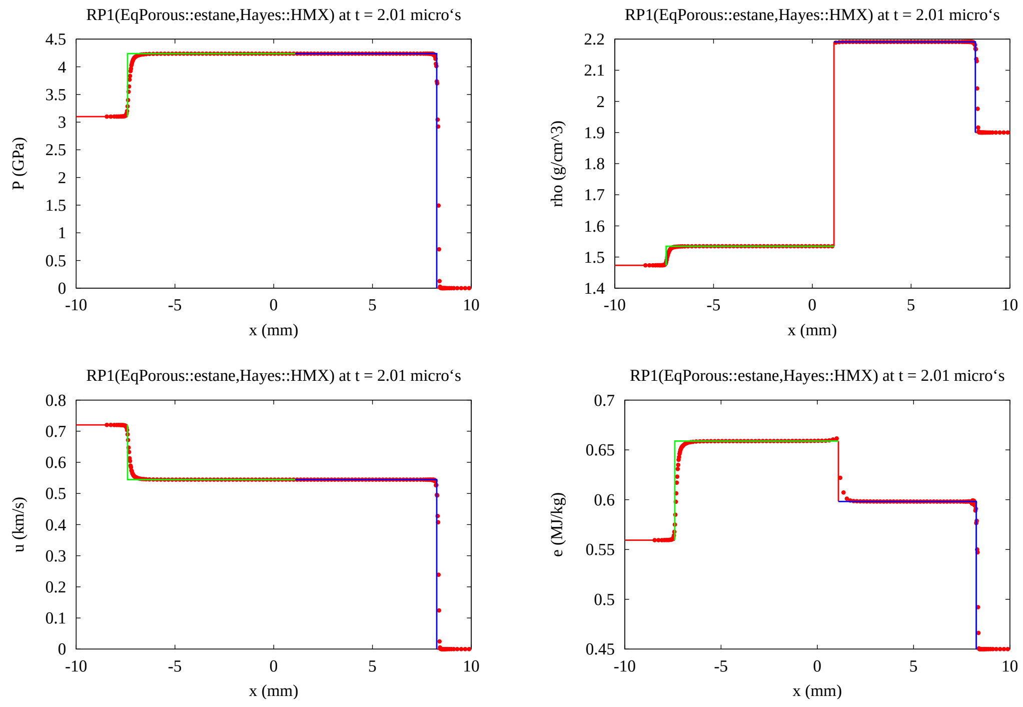


Figure 3: Comparison of numerical profiles and exact solution for Riemann problem with two outgoing shocks. Simulation uses mesh refinement.

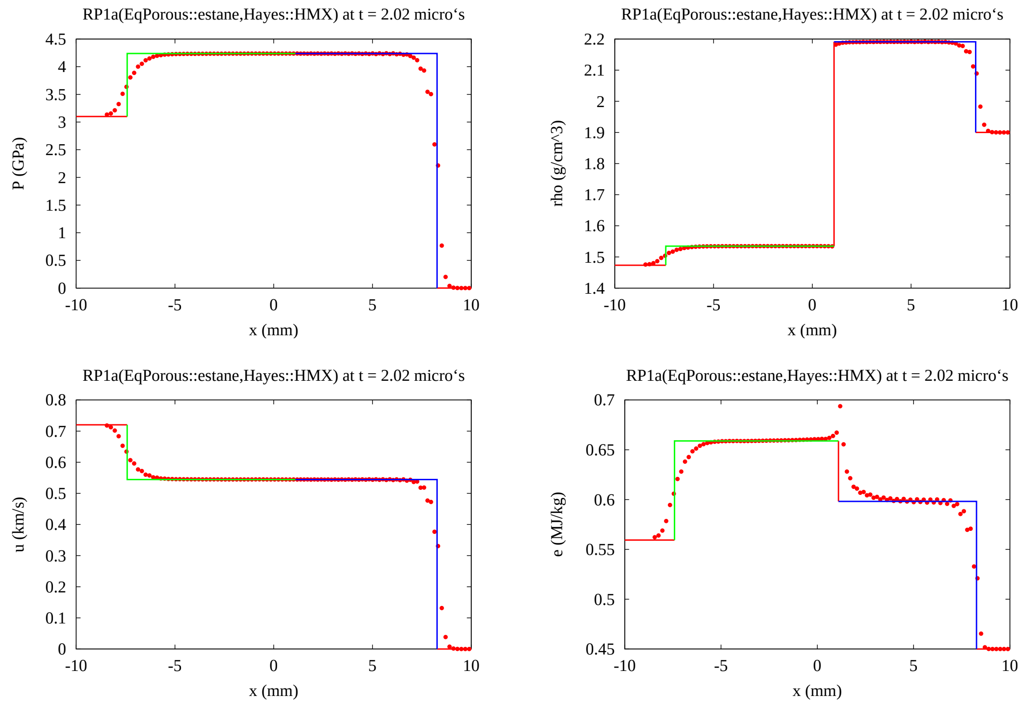


Figure 4: Comparison of numerical profiles and exact solution for Riemann problem with two outgoing shocks. Simulation did not use mesh refinement.

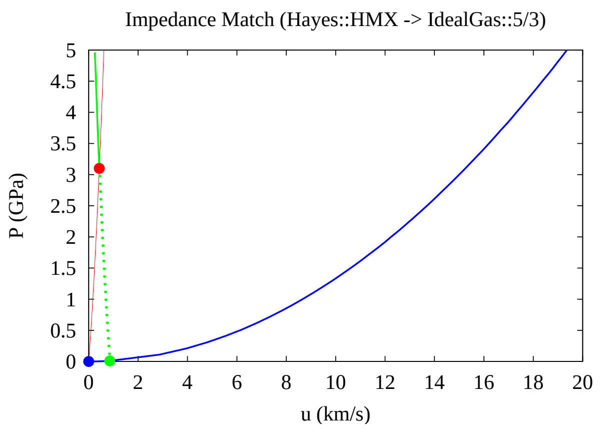


Figure 5: Graphical solution to second impedance match problem which has an outgoing rarefaction wave and shock wave.

degeneracy of the mode with characteric velocity 0, the system of the PDEs remains hyperbolic with one pair of acoustic modes. The additional degrees of freedom $\vec{Z}$ are coupled to the fluid flow by means of the pressure $P_n(V, e, \vec{Z})$ in the flux function, and perhaps a source term $Q(V, e, \vec{Z})$ in the energy equation.

For reactive flow, the internal degrees of freedom are typically the mass fractions of the reactants and $\vec{R}$ represents the chemical reaction rates. Models for non-equilibrium effects in gasdynamics, such as dissociation, ionization and vibrational or rotational excitations, can be viewed as a special case of reactive flow.

For elastic-plastic flow, P_n corresponds to the longitudinal component of stress σ_{xx}. The internal variables are typically identified with plastic strain, work hardening parameter, back stress, *etc*. The source term for the plastic strain is known as the plastic-strain rate, and typically vanishes when the shear stress is below the yield surface. This is similar to chemical reaction rates decreasing rapidly with temperature and freezing in non-equilibium species, such as occurs in the exhaust nozzle of a jet engine.

The material properties for this class of models are specified by a vector of functions (P_n, Q, $\vec{R}$). As with the composite equations of state in section 2.3, von Mises Elastic-Plastic EOS and equilibrium porous EOS, these functions can be considered to be virutal functions of a materials property class. Again the structure used for the EOS package, consisting of a database file for parameters and shared libraries to implement specific functions, would allow different applications to utilize a common set of models for a wide variety of materials.

Furthermore, different physics models can easily be combined within the same continuum mechanics code. Suppose an application consists of an explosive (reactive material) driving a metal plate (elastic-plastic material) through a gas (hydrostatic material). A simulation code would have to allocate enough components in its solution vector for the maximum number of internal degrees of freedom of any material being used. Then by using the virtual functions in the material class to evaluate the source terms, the components of the solution vector for each cell would be updated with the appropriate material model. In addition, different solvers can be used for each region provided that the flux at the interface between regions is consistent. For example, specialized solvers could be used in regions with stiff source terms. With this framework, the overall efficiency of a simulation could be improved with a hybrid algorithm that selects a solver well suited to the physics within each region.

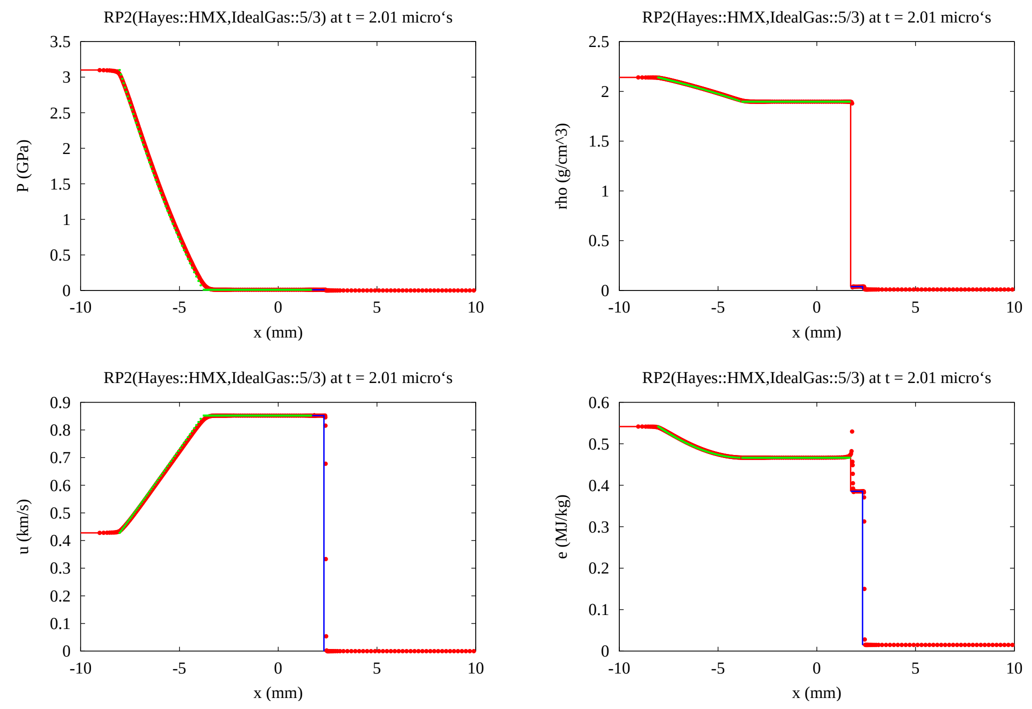


Figure 6: Comparison of numerical profiles and exact solution for Riemann problem with outgoing rarefaction wave and shock wave. Simulation uses mesh refinement.

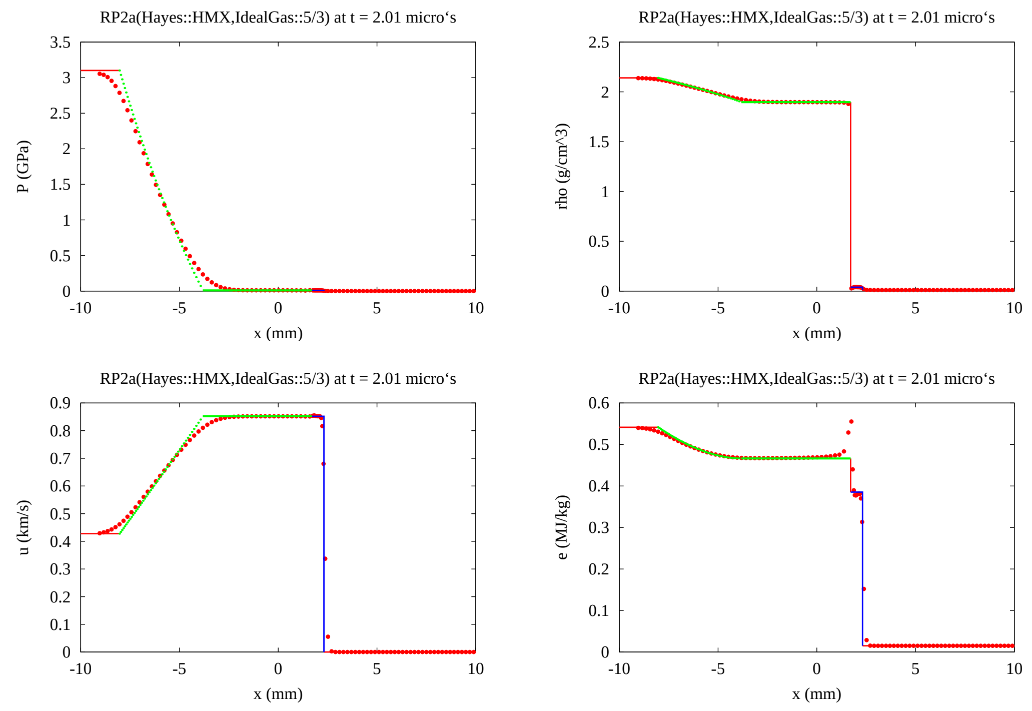


Figure 7: Comparison of numerical profiles and exact solution for Riemann problem with outgoing rarefaction wave and shock wave. Simulation did not use mesh refinement.

6 Final Remarks

The utility and effectiveness of software, such as an EOS package, depends on the details of the implementation as well as the overall structure. Yet details in an article such as this is necessarily limited by constraints on the length and the fact that only very few specialist have the inclination or motivation to analyze an implementation of another researcher. Reimplementation of the concepts and ideas results in considerable duplication of effort and slows down progress towards improving the software.

Software is especially important to the computational fluid dynamics (CFD) community. A large number of algorithms have been proposed. Yet for compressible flow, the strong non-linearities have limited analysis mostly to a scalar PDE in one-dimension. Algorithms for systems of PDEs are typically based on a heuristic derivation motivated by the scalar case. Moreover, characteristic analysis, that is such a powerful technique in one-dimension, is rather limited in higher dimensions. Typically, algorithms in two and three dimensions are based on operator splitting. Consequently, algorithmic development relies heavily on numerical experiments.

Yet, the CFD community has not agreed upon a standard set of test problems. Agreement is hindered by the time and effort required to setup numerical problems. Morover, the time and effort to code up different algorithms, along with their many variations, has prevented a systematic comparisons between algorithms. This has greatly slowed down progress in the field. One example is the ongoing controversity, after thirty years of experiments and simulations, about the ‘transition criterion’ for regular and Mach reflections. Another example is that of the four solvers tried with the EOS package presented here, one was significantly worse than the others. Is this because the scheme was developed and tested on an Eulerian grid and not suited to a Lagrangian grid, or that some part of the statement of the algorithm is ambiguous and has been misinterpreted, or that the algorithm is coded incorrectly? Having a mechanism to exchange source code would greatly faciliate the ability to track down discrepancies between claims for an algorithm in the literature and the experience of an individual researcher on a particular application.

The CFD has reached a level of maturity where it is not practical for every research group to produce all its own software from scratch. In analogy with standard mathematical libraries for special functions and linear algebra, the CFD community needs a means to share the components of a hydro algorithm and have them available at the level of source code. Sharing code will require standards for arguments to functions and protocols for how the components interact. This is quiet analogous to protocols like TCP/IP which allowed local networks to expand into the world wide web and language standards like html which enabled web browsers to become available to all PC users.

We note that the solvers and scripts discussed here can be combined in a tar file of about 100 Kbytes, and then easily sent by e-mail or made available over the WEB.[1] With this file, and with Amrita and the EOS package preinstalled, all the results presented here, and in fact this manuscript, can be reproduced. Making available research at the level of code would allow other researchers readily to try out their own conjectures without the large overhead in time and effort of recoding and debuging a published algorithm. The ability for anyone to run a suite of simulations would facilitate determining the strengths and weakness of an algorithm and allow subjective claims about the merits of one algorithm as compared to another to be analyzed in an objective manner. This would naturally lead to increasing standards in a field that relies heavily on numerical simulations.

The Amrita environment can be thought of as a proof-of-principle that a flexible and extendible software environment for hydrodynamics can be designed. The use of a high level script language tuned to fluid dynamics applications and plugins for solver algorithms and specialized code, such as a material property library, would go a long way to alleviate the obstacle to more rapid progress. This approach requires consensus of the CFD community and a change from the cottage industry mentality, that developed when

[1]The EOS library presented here is available from `http://t14web.lanl.gov/Staff/rsm/preprints.html#EOSlib`.

computing resources were limited to those few employed at large research centers, to a more cooperative mode in which code is shared, examined by many researchers and continually improved for the benefit of the entire community. A cooperative approach requires standardization in order for software components to be interchangable.

References

[1] R. Donat and A. Marquina, *Capturing shock reflections: An improved flux formula*, J. Comput. Phys. **125** (1996), 42–58. 4.1

[2] P. Glaister, *An aproximate linearized Riemann solver for the Euler equations of real gases*, J. Comput. Phys. **74** (1988), 382–408. 4.1

[3] ———, *A Riemann solver for compressible flow of a real gas in a lagrangian frame*, Computers Math. Applic. **22** (1991), 35–39. 4.1

[4] J. N. Johnson, D. B. Hayes, and J. R. Asay, *Equations of state and shock-induced transformations in solid i-solid ii-liquid bismuth*, J. Phys. Chem. Solids. **35** (1974), 501–515. 3

[5] S. P. Marsh (ed.), *LASL shock Hugoniot data*, Univ. of Calif. Press., 1980, `http://lib-www.lanl.gov/books/shd.pdf`. 3

[6] R. Menikoff, *Errors when shock waves interact due to numerical shock width*, SIAM J. SCI. Comput. **15** (1994), 1227–1242. 4.3

[7] R. Menikoff and E. Kober, *Equation of state and Hugoniot locus for porous materials: P–α model revisited*, APS Topical Conference of Shock Compression in Condensed Matter–1999, 1999. 5

[8] R. Menikoff and B. J. Plohr, *The Riemann problem for fluid flow of real materials*, Revs. Mod. Phys. **61** (1989), 75–130. 1

[9] S. Muller and A. Voss, *A Riemann solver for the Euler equations with non-convex equation of state*, Tech. Report IGPM-197, RWTH-Aachen, Germany, 1999, `http://www.igpm.rwth-aachen.de/~voss/research_papers.html`. 2.2

[10] W. F. Noh, *Errors for calculations of strong shocks using an artificial viscosity and an artificial heat flux*, J. Comp. Phys. **72** (1987), 78–120. 4.3

[11] J. J. Quirk, *Amrita - a computational facility for cfd modelling*, p. ?, von Karmen Institute, 1998, `http://www.galcit.caltech.edu/~jjq/vki/vki:cfd29::jjq_l1.ps.gz`. 1, 4

[12] ———, *Amr_sol: Design principles and practice*, p. ?, von Karmen Institute, 1998, `http://www.galcit.caltech.edu/~jjq/vki/vki:cfd29::jjq_l2.ps.gz`. 1, 4.1

[13] P. L. Roe, *Approximate Riemann solvers, parameter vectors, and difference-schemes*, J. Comput. Phys. **43** (1981), 357–372. 4.1

[14] G. Sod, *A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws*, J. Comput. Phys. **27** (1978), 1–31. 4.3